
A Logic-based Approach to Generatively Defined Discriminative Modeling

Taisuke Sato · Keiichi Kubota · Yoshitaka Kameya

Oct. 2014

Abstract Conditional random fields (CRFs) are usually specified by graphical models but in this paper we propose to use probabilistic logic programs and specify them generatively. Our intension is first to provide a unified approach to CRFs for complex modeling through the use of a Turing complete language and second to offer a convenient way of realizing generative-discriminative pairs in machine learning to compare generative and discriminative models and choose the best model. We implemented our approach as the D-PRISM language by modifying PRISM, a logic-based probabilistic modeling language for generative modeling, while exploiting its dynamic programming mechanism for efficient probability computation. We tested D-PRISM with logistic regression, a linear-chain CRF and a CRF-CFG and empirically confirmed their excellent discriminative performance compared to their generative counterparts, i.e. naive Bayes, an HMM and a PCFG. We also introduced new CRF models, CRF-BNCs and CRF-LCGs. They are CRF versions of Bayesian network classifiers and probabilistic left-corner grammars respectively and easily implementable in D-PRISM. We empirically showed that they outperform their generative counterparts as expected.

Keywords CRFs · D-PRISM · logic-based

Taisuke Sato
Tokyo Institute of Technology, Japan
Tel.: +81-3-5743-2186
E-mail: sato@mi.cs.titech.ac.jp

Keiichi Kubota
Tokyo Institute of Technology, Japan
Tel.: +81-3-5743-2186
E-mail: kubota@mi.cs.titech.ac.jp

Yoshitaka Kameya
Meijo University, Japan
Tel. : +81-52-838-2567
E-mail: ykameya@meijo-u.ac.jp

1 Introduction

Conditional random fields (CRFs) [11] are probabilistic models for discriminative modeling defining a conditional distribution $p(y \mid x)$ over output y given input x . They are quite popular for labeling sequence data such as text data and biological sequences [26]. Although they are usually specified by graphical models, we here propose to use probabilistic logic programs and specify them generatively. Our intension is first to provide a unified approach to CRFs for complex modeling through the use of a Turing complete language and second to offer a convenient way of realizing generative-discriminative pairs [19] in machine learning to compare generative and discriminative models and choose the best model.

The use of logical expressions to specify CRFs is not new but they have been used solely as feature functions [8, 21]. For example in Markov logic networks (MLNs) [21], weighted clauses are used as feature functions to define (conditional) Markov random fields and probabilities are obtained by Gibbs sampling. In contrast, our approach is implemented by a generative modeling language PRISM [22, 23] where clauses have no weights; they simply constitute a logic program DB computing possible output y from input x by proving a top-goal $G_{x,y}$ that relates x to y . In addition probabilities are exactly computed by dynamic programming. DB however contains special atoms of the form $\text{msw}(i, v)$ having weights $\exp(\lambda_{i,v})$ where i and v are arbitrary terms. They are called **msw** atoms here as in PRISM. We define the weight $q(x, y)$ of a top-goal $G_{x,y}$ as a sum-product of such weights associated with **msw** atoms appearing in a proof of $G_{x,y}$ and consider $q(x, y)$ as an unnormalized distribution. By modifying the dynamic programming mechanism of PRISM slightly, we can efficiently compute, when possible and feasible, the unnormalized marginal distribution $q(x) = \sum_y q(x, y)$ and obtain a CRF $p(y \mid x) = q(x, y)/q(x)$. We implemented our idea by modifying PRISM and termed the resulting language D-PRISM (discriminative PRISM). D-PRISM is a general programming language that generatively defines CRFs and provides built-in predicates for parameter learning and Viterbi inference of CRFs.

Our approach to CRFs is general in the sense that, like other statistical relational learning (SRL) languages for CRFs [21, 17], programs in D-PRISM have no restriction such as the exclusiveness condition in PRISM [23] except for the use of binary features and we can write any program, i.e. we can write arbitrary CRFs as long as they are described by D-PRISM. We point out that binary features are the most common features and they can encode basic CRF models such as logistic regression, linear-chain CRFs and CRF-CFGs [9, 5, 26]. Furthermore by dynamic programming, probabilistic inference can be efficiently carried out with the same time complexity as their generative counterparts as exemplified by linear-chain CRFs and hidden Markov models (HMMs).

In machine learning it is well-known that naive Bayes and logistic regression form a generative-discriminative pair [19]. That is, any conditional distribution $p(y \mid \mathbf{x})$ computed from a joint distribution $p(\mathbf{x}, y) = p(\mathbf{x} \mid y)p(y)$ defined generatively by naive Bayes, where y is a class and \mathbf{x} is a feature vector, can also be defined directly by logistic regression and vice versa. As is empirically demonstrated in [19], classification accuracy by discriminative models such as logistic regression is generally better than their corresponding generative models such as naive Bayes when there is enough data but generative models reach their best performance more quickly than discriminative ones w.r.t. the amount of available

data. Also the theoretical analysis in [12] suggests that when a model is wrong in generative modeling, the deterioration of prediction accuracy is more severe than in discriminative modeling. It seems therefore reasonable to say “...For any particular data set, it is impossible to predict in advance whether a generative or a discriminative model will perform better” [26]. Hence what is desirable is to provide a modeling environment in which the user can test both types of modeling smoothly without pain and D-PRISM provides such an environment that makes it easy to test and compare discriminative modeling and generative modeling for the same class or related family of probabilistic models.

In what follows, we review CRFs in Section 2 and also review three basic models, i.e. logistic regression, linear-chain CRFs and CRF-CFGs in Section 3. We then introduce D-PRISM in Section 4. We empirically verify the effectiveness of our approach in Section 5 using the three models. Section 6 introduces new CRF models, CRF-BNCs and CRF-LCGs, both easily implementable in D-PRISM. In Section 7, we discuss program transformation which derives a program for incomplete data from one for complete data. Section 8 contains related work and discussion and Section 9 is the conclusion.

2 Conditional random fields

Conditional random fields (CRFs) [11] are popular probabilistic models defining a conditional distribution $p(\mathbf{y} \mid \mathbf{x})$ over the output sequence \mathbf{y} given an input sequence \mathbf{x} which takes the following form¹:

$$p(\mathbf{y} \mid \mathbf{x}) \equiv \frac{1}{Z(\mathbf{x})} \exp \left\{ \sum_{k=1}^K \lambda_k f_k(\mathbf{x}, \mathbf{y}) \right\}.$$

Here $f_k(\mathbf{x}, \mathbf{y})$ and λ_k ($1 \leq k \leq K$) are respectively a real valued function (*feature function*) and the associated weight (*parameter*) and $Z(\mathbf{x})$ a normalizing constant. As $Z(\mathbf{x})$ is the sum of exponentially many terms, the exact computation is generally intractable and takes $\mathcal{O}(M^{|\mathbf{y}|})$ time where M is the maximum number of possible values for each component of \mathbf{y} and hence approximation methods have been developed [26]. However when $p(\mathbf{y} \mid \mathbf{x})$ has recursive structure of specific type as a graphical model like linear-chain CRFs, $Z(\mathbf{x})$ is efficiently computable by dynamic programming.

Now let $D = \{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), \dots, (\mathbf{x}^{(T)}, \mathbf{y}^{(T)})\}$ be a training set. The regularised conditional log-likelihood $l(\boldsymbol{\lambda} \mid D)$ of D is given by

$$\begin{aligned} l(\boldsymbol{\lambda} \mid D) &\equiv \sum_{t=1}^T \log p(\mathbf{y}^{(t)} \mid \mathbf{x}^{(t)}) - \frac{\mu}{2} \sum_{k=1}^K \lambda_k^2 \\ &= \sum_{t=1}^T \left\{ \sum_{k=1}^K \lambda_k f_k(\mathbf{x}^{(t)}, \mathbf{y}^{(t)}) - \log Z(\mathbf{x}^{(t)}) \right\} - \frac{\mu}{2} \sum_{k=1}^K \lambda_k^2 \end{aligned}$$

¹ Bold italic letters are (values of) random vectors in this paper.

where $\boldsymbol{\lambda} = \lambda_1, \dots, \lambda_K$ are parameters and $\frac{\mu}{2} \sum_{k=1}^K \lambda_k^2$ is a penalty term. Parameters are then estimated as the ones that maximize $l(\boldsymbol{\lambda} \mid D)$ by Newton's method or quasi-Newton methods. The gradient required for parameter learning is computed as

$$\frac{\partial l(\boldsymbol{\lambda} \mid D)}{\partial \lambda_k} = \sum_{t=1}^T \left\{ f_k(\mathbf{x}^{(t)}, \mathbf{y}^{(t)}) - E(f_k \mid \mathbf{x}^{(t)}) \right\} - \mu \lambda_k.$$

The problem here is that the expectation $E(f_k \mid \mathbf{x}^{(t)})$ is difficult to compute and hence a variety of approximation methods such as stochastic gradient descent (SDG) [26] have been proposed. However in this paper we focus on cases where exact computation by dynamic programming is possible and use an algorithm that generalizes inside probability computation in probabilistic context free grammars (PCFGs) [15].

After parameter learning, we apply our model to prediction tasks and infer the most-likely output $\hat{\mathbf{y}}$ for an input sequence \mathbf{x} using

$$\begin{aligned} \hat{\mathbf{y}} &\equiv \operatorname{argmax}_{\mathbf{y}} p(\mathbf{y} \mid \mathbf{x}) \\ &= \operatorname{argmax}_{\mathbf{y}} \frac{1}{Z(\mathbf{x})} \exp \left\{ \sum_{k=1}^K \lambda_k f_k(\mathbf{x}, \mathbf{y}) \right\} = \operatorname{argmax}_{\mathbf{y}} \sum_{k=1}^K \lambda_k f_k(\mathbf{x}, \mathbf{y}). \end{aligned}$$

As naively computing $\hat{\mathbf{y}}$ is straightforward but too costly, we again consider only cases where dynamic programming is feasible and apply a variant of the Viterbi algorithm for HMMs.

3 Basic models

3.1 Logistic regression

Logistic regression specifies a conditional distribution $p(y \mid \mathbf{x})$ over a class variable y given the input $\mathbf{x} = x_1, \dots, x_K$, a vector of attributes. It assumes $\log p(y \mid \mathbf{x})$ is a linear function of \mathbf{x} and given by

$$p(y \mid \mathbf{x}) \equiv \frac{1}{Z(\mathbf{x})} \exp \left\{ \lambda_y + \sum_{j=1}^K \lambda_{y,j} x_j \right\}.$$

We here confirm that logistic regression is a CRF. Rewrite $\lambda_y = \sum_{y'} \lambda_{y'} \mathbf{1}_{\{y'=y\}}$ and $\lambda_{y,j} x_j = \sum_{y'} \lambda_{y',j} \mathbf{1}_{\{y'=y\}} x_j$ and substitute them for λ_y and $\lambda_{y,j} x_j$ in the above formula². We obtain

$$p(y \mid \mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp \left\{ \sum_{y'} \lambda_{y'} \mathbf{1}_{\{y'=y\}} + \sum_{y'} \sum_{j=1}^K \lambda_{y',j} \mathbf{1}_{\{y'=y\}} x_j \right\}.$$

² $\mathbf{1}_{\{y'=y\}}$ is a binary function of y taking 1 if $y = y'$, otherwise 0.

By considering $\mathbf{1}_{\{y'=y\}}$ and $\mathbf{1}_{\{y'=y\}}x_j$ as feature functions (of y and \mathbf{x}), we can see logistic regression is a CRF.

3.2 Linear-chain CRFs

CRFs [11] are generally intractable and a variety of approximation methods such as sampling and loopy BP have been developed. There is however a tractable subclass called *linear-chain CRFs*. They are of the following the form:

$$p(\mathbf{y} \mid \mathbf{x}) \equiv \frac{1}{Z(\mathbf{x})} \exp \left\{ \sum_{k=1}^K \lambda_k \sum_{i=2}^N f_k(\mathbf{x}, y_i, y_{i-1}) \right\}$$

where $Z(\mathbf{x})$ is a normalizing constant. They define, as CRFs, a conditional distribution $p(\mathbf{y} \mid \mathbf{x})$ over output sequences \mathbf{y} given an input sequence \mathbf{x} such that $|\mathbf{x}| = |\mathbf{y}| = N$ ($|\mathbf{x}|$ denotes the length of vector \mathbf{x}) but feature functions are restricted to the form $f(\mathbf{x}, y_i, y_{i-1})$ ($\mathbf{y} = y_1, \dots, y_N, 2 \leq i \leq N$) which only refers to two consecutive components in \mathbf{y} . Thanks to this local reference restriction exact probability computation is possible for linear-chain CRFs in time linear in the input length $|\mathbf{x}|$ by a variant of the forward-backward algorithm for HMMs. Linear-chain CRFs are considered as a generalized and undirected version of HMMs which enable us to use far richer feature functions other than transition probabilities and emission probabilities used in HMMs.

3.3 CRF-CFGs

PCFGs [15] are a basic class of probabilistic grammars extending CFGs by assigning selection probabilities θ to production rules. In PCFGs, the probability of a sentence is the sum of probabilities of parse trees and the probability of a parse tree is the product of probabilities associated with production rules used in the tree. PCFGs are generative models and parameters are usually learned by maximum likelihood estimation (MLE). So given parse trees τ_1, \dots, τ_T and the corresponding sentences s_1, \dots, s_T , parameters are estimated as $\theta^* = \operatorname{argmax}_{\theta} \prod_{t=1}^T p(\tau_t, s_t \mid \theta)$.

Seeking better parsing accuracy, Johnson attempted parameter learning by maximizing conditional likelihood: $\theta^\dagger = \operatorname{argmax}_{\theta} \prod_{t=1}^T p(\tau_t \mid s_t, \theta)$ but found the improvement is not statistically significant [9]. Later Finkel et al. generalized PCFGs to *conditional random field context free grammars* (CRF-CFGs) [5] where the conditional probability $p(\tau \mid s)$ of a parse tree τ given a sentence s is defined by

$$p(\tau \mid s) \equiv \frac{1}{Z(s)} \exp \left\{ \sum_{k=1}^K \lambda_k \sum_{r \in \tau} f_k(r, s) \right\}.$$

Here $Z(s)$ is a normalizing constant. $\lambda_1, \dots, \lambda_K$ are parameters and $r \in \tau$ is a CFG rule (possibly enriched with other information) appearing in the parse tree τ of s and $f_k(r, s)$ is a feature function. Finkel et al. conducted learning experiments with a CRF-CFG using the Penn Treebank [16]. They learned parameters from

parse trees τ_1, \dots, τ_T and the corresponding sentences s_1, \dots, s_T in the corpus by maximizing conditional likelihood just like [9] but this time they obtained a significant gain in parsing accuracy [5]. Their experiments clearly demonstrate the advantage of extensive use of features and discriminative parameter learning.

4 D-PRISM

Having seen basic models of CRFs, we next show how they are uniformly subsumed by a logic-based modeling language PRISM [22,23] with a simple modification of its probability computation. The modified language is termed *D-PRISM* (discriminative PRISM).

4.1 PRISM at a glance

Before proceeding we quickly review PRISM³. PRISM is a high-level generative modeling language based on Prolog, extended by a rich array of probabilistic built-in predicates for various types of probabilistic inference and parameter learning. Specifically it offers, in addition to MLE by the EM algorithm, Viterbi training (VT), variational Bayes (VB), variational VT (VB-VT) and MCMC for Bayesian inference. PRISM has been applied to music and bioinformatics [25,1,18].

Syntactically a PRISM program *DB* is a Prolog program and runs like Prolog. Fig. 1 is an example of PRISM program for naive Bayes. *DB* is basically a set of definite clauses. The difference from usual Prolog programs is that the clause body may contain special atoms, **msw** atoms⁴, of the form **msw**(*i*, *v*) representing a probabilistic choice made by (analogically speaking) rolling a die *i* and choosing the outcome *v*. Here *i* is a ground term naming the **msw** atom and *v* belongs to a set V_i of possible outcomes declared by **values/2** declaration. The probability of **msw**(*i*, *v*) ($v \in V_i$) being true is denoted by $\theta_{i,v}$ and called a *parameter* for **msw**(*i*, *v*). Naturally $\sum_{v \in V_i} \theta_{i,v} = 1$ holds. Executing **msw**(*i*, *X*) with a variable *X* returns a value *v* $\in V_i$ in *X* with probability $\theta_{i,v}$. So **msw**(season, *S*) in Fig. 1 probabilistically returns one of {spring, summer, fall, winter} in *S*.

```

values(season,[spring,summer,fall,winter]).
values(attr(temp,_),[high,mild,low]).
values(attr(humidity,_),[high,low]).

nb([T,H],S):-
    msw(season,S),           % defines p(X,Y) where X = [T,H] and Y = S
    msw(attr(temp,S),T),     % choose S from {spring,summer,fall,winter}
    msw(attr(humidity,S),H). % choose T from {high,mild,low}
                             % choose H from {high,low}
nb([T,H]):- nb([T,H],_).    % defines p(X) where X = [T,H]
```

Fig. 1 PRISM program *DB*₀ for naive Bayes

³ <http://sato-www.cs.titech.ac.jp/prism/>

⁴ **msw** stands for “multi-valued switch.”

DB defines a probability measure $p_{DB}(\cdot)$ over Herbrand interpretations (possible worlds)[23]. The probability $p_{DB}(G)$ of a top-goal G then is computed as a sum-product of parameters in two steps. First G is reduced using DB by SLD search to a disjunction $E_1 \vee \dots \vee E_M$ such that each E_j ($1 \leq j \leq M$) is a conjunction of **msw** atoms representing a sequence of probabilistic choices. E_j is called an *explanation* for G because it explains why G is true or how G is proved as a result of probabilistic choices encoded by E_j . Let $\phi(G) \equiv \{E_1, \dots, E_M\}$ be the set of all explanations for G . $p_{DB}(G)$ is computed as $p_{DB}(G) = \sum_{E \in \phi(G)} p_{DB}(E)$ and $p_{DB}(E) = \prod_{k=1}^N \theta_k$ for $E = \mathbf{msw}_1 \wedge \dots \wedge \mathbf{msw}_N$, where θ_k is a parameter for \mathbf{msw}_k ($1 \leq k \leq N$).

Let $p(x, y)$ be a joint distribution over input x (or observation) and output y (or hidden state) which we wish to compute by a PRISM program. We write a program DB that probabilistically proves $G_{x,y}$, a top-goal that relates x to y , using **msw** atoms, in such a way that $p(x, y) = p_{DB}(G_{x,y})$ holds. Since (x, y) forms complete data, $G_{x,y}$ has only one explanation $E_{x,y}$ for $G_{x,y}$ ⁵, so we have $p(x, y) = p_{DB}(G_{x,y}) = p_{DB}(E_{x,y}) = \prod_{i,v} \theta_{i,v}^{\sigma_{i,v}(E_{x,y})}$ where $\sigma_{i,v}(E_{x,y})$ is the count of $\mathbf{msw}(i, v)$ in $E_{x,y}$. Introduce $G_x = \exists y G_{x,y}$. Then the marginal probability $p(x)$ is obtained as $p_{DB}(G_x)$ because $p(x) = \sum_y p(x, y) = \sum_y p_{DB}(G_{x,y}) = p_{DB}(G_x)$ holds. Hence the conditional distribution $p(y | x)$ is computed as

$$p(y | x) = \frac{p_{DB}(G_{x,y})}{p_{DB}(G_x)} = \frac{p_{DB}(E_{x,y})}{p_{DB}(G_x)} = \frac{\prod_{i,v} \theta_{i,v}^{\sigma_{i,v}(E_{x,y})}}{\sum_{E_{x,y} \in \phi(G_x)} \prod_{i,v} \theta_{i,v}^{\sigma_{i,v}(E_{x,y})}}. \quad (1)$$

We next apply (1) to the naive Bayes program DB_0 in Fig. 1. DB_0 is intended to infer a season **S** from temperature **T** and humidity **H** and generatively defines a joint distribution $p([\mathbf{T}, \mathbf{H}], \mathbf{S}) = p_{DB_0}(\mathbf{nb}([\mathbf{T}, \mathbf{H}], \mathbf{S}))$. To draw a sample from $p([\mathbf{T}, \mathbf{H}], \mathbf{S})$ or equivalently from $p_{DB_0}(\mathbf{nb}([\mathbf{T}, \mathbf{H}], \mathbf{S}))$, it first samples a season **S** by executing **msw(season, S)**, then similarly samples a value **T** of temperature and a value **H** of humidity, each conditioned on **S**, by executing **msw(attr(temp, S), T)** and **msw(attr(humidity, S), H)** in turn⁶. Note that the program also includes a clause **nb([T, H]):-nb([T, H], _)** to compute a marginal distribution $p_{DB_0}(\mathbf{nb}([\mathbf{T}, \mathbf{H}]))$ ($= p([\mathbf{T}, \mathbf{H}])$). The correspondence to (1) is that $x = [\mathbf{T}, \mathbf{H}]$, $y = \mathbf{S}$, $G_{x,y} = \mathbf{nb}([\mathbf{T}, \mathbf{H}], \mathbf{S})$ and $G_x = \mathbf{nb}([\mathbf{T}, \mathbf{H}])$. The conditional distribution $p(\mathbf{S} | [\mathbf{T}, \mathbf{H}])$ is computed as $p_{DB_0}(\mathbf{nb}([\mathbf{T}, \mathbf{H}], \mathbf{S})) / p_{DB_0}(\mathbf{nb}([\mathbf{T}, \mathbf{H}]))$.

4.2 From probability to weight

The basic idea of our approach to discriminative modeling is to generalize (1) by replacing probability $\theta_{i,v}$ for $\mathbf{msw}(i, v)$ with arbitrary *weight* $\eta_{i,v} = \exp(\lambda_{i,v})$. In D-PRISM we further perform normalization to obtain a CRF. More precisely, we first introduce an *unnormalized distribution* $q(x, y) = q_{DB}(G_{x,y})$ defined by:

$$q_{DB}(G_{x,y}) \equiv \exp\left(\sum_{i,v} \lambda_{i,v} \sigma_{i,v}(E_{x,y})\right)$$

where $E_{x,y}$ is a unique explanation for $G_{x,y}$

⁵ This is an assumption but generally true with programs for complete data.

⁶ For example **msw(attr(temp, S), T)** samples **T** from the conditional distribution $p(\mathbf{T} | \mathbf{S})$.

assuming that for any complete data (x, y) and the corresponding top-goal $G_{x,y}$, our program, DB , always has only one explanation $E_{x,y}$. By setting $\lambda_{i,v} = \ln \theta_{i,v}$, $q_{DB}(G_{x,y})$ is reduced to $p_{DB}(G_{x,y})$ again.

Next we rewrite (1) as follows by putting $p(E_{x,y} \mid G_x) \equiv \frac{q_{DB}(G_{x,y})}{\sum_y q_{DB}(G_{x,y})}$ and using $\eta_{i,v} = \exp(\lambda_{i,v})$.

$$p(E_{x,y} \mid G_x) = \frac{1}{Z(G_x)} \exp\left(\sum_{i,v} \lambda_{i,v} \sigma_{i,v}(E_{x,y})\right) = \frac{1}{Z(G_x)} \prod_{i,v} \eta_{i,v}^{\sigma_{i,v}(E_{x,y})} \quad (2)$$

$$\begin{aligned} Z(G_x) &= \sum_{E_{x,y} \in \phi(G_x)} \exp\left(\sum_{i,v} \lambda_{i,v} \sigma_{i,v}(E_{x,y})\right) \\ &= \sum_{E_{x,y} \in \phi(G_x)} \prod_{i,v} \eta_{i,v}^{\sigma_{i,v}(E_{x,y})} \end{aligned} \quad (3)$$

(2) and (3) are fundamental equations for D-PRISM describing how a CRF $p(y \mid x) = p(E_{x,y} \mid G_x)$ is defined and computed. By comparing (1) to (2) and (3), we notice that the most computationally demanding task in D-PRISM, computing $Z(G_x)$ in (3), can be carried out efficiently by dynamic programming just by replacing probability $\theta_{i,v}$ in PRISM with weight $\eta_{i,v}$, resulting in the same time complexity for probability computation as in PRISM.

It is also seen from (2) and (3) that in our formulation of CRFs by D-PRISM, $\sigma_{i,v}(E_{x,y})$, the count of $\mathbf{msw}(i, v)$ in $E_{x,y}$, works as a (default) feature function⁷ over the input x and output y . $\sigma_{i,v}(E_{x,y})$ becomes binary when $\mathbf{msw}(i, v)$ occurs at most once in $E_{x,y}$. For a binary feature function $f(x, y)$ in general, let $\mathbf{msw}(\mathbf{f}(x, y), 1)$ be a dummy \mathbf{msw} atom which is unique to $f(x, y)$ and always true. We assume that corresponding to $f(x, y)$, there is a goal $\mathbf{f}(x, y)$ provable in PRISM if and only if $f(x, y) = 1$. Then it is easy to see that a PRISM goal $(\mathbf{f}(x, y) \rightarrow \mathbf{msw}(\mathbf{f}(x, y), 1) ; \mathbf{true})$ realizes $f(x, y)$.

From the viewpoint of modeling, we emphasize that for the user, D-PRISM programs are just PRISM programs that proves two top-goals, $G_{x,y}$ for complete data (x, y) and G_x for incomplete data x . For example, the PRISM program in Fig. 1 for naive Bayes is also a D-PRISM program defining logistic regression.

In D-PRISM, parameters are learned discriminatively from complete data. Consider the regularised (log) conditional likelihood $l(\lambda \mid D)$ of a set of observed data $D = \{d_1, d_2, \dots, d_T\}$ where $d_t = (G_{x(t)}, E_{x(t), y(t)}) = (G_t, E_t)$ ($1 \leq t \leq T$). $l(\lambda \mid D)$ is given by

$$\begin{aligned} l(\lambda \mid D) &\equiv \sum_{t=1}^T \log p(E_t \mid G_t) - \frac{\mu}{2} \sum_{i,v} \lambda_{i,v}^2 \\ &= \sum_{t=1}^T \left\{ \sum_{i,v} \lambda_{i,v} \sigma_{i,v}(E_t) - \log Z(G_t) \right\} - \frac{\mu}{2} \sum_{i,v} \lambda_{i,v}^2 \end{aligned}$$

⁷ Since we assume that the top-goal $G_{x,y}$ has only one explanation $E_{x,y}$ for a complete data (x, y) , (x, y) uniquely determines $\sigma_{i,v}(E_{x,y})$.

and parameters $\lambda = \{\lambda_{i,v}\}$ are determined as the ones that maximize $l(\lambda \mid D)$. Currently we use L-BFGS [13] to maximize $l(\lambda \mid D)$. The gradient used in the maximization is computed as

$$\begin{aligned} \frac{\partial l(\lambda \mid D)}{\partial \lambda_{i,v}} &= \sum_{t=1}^T \left\{ \sigma_{i,v}(E_t) - \frac{\partial}{\partial \lambda_{i,v}} \log Z(G_t) \right\} - \mu \lambda_{i,v} \\ &= \sum_{t=1}^T \left\{ \sigma_{i,v}(E_t) - \sum_{E' \in \phi(G_t)} \sigma_{i,v}(E') p(E' \mid G_t) \right\} - \mu \lambda_{i,v}. \end{aligned}$$

Finally, Viterbi inference, computing the most likely output y for the input x , or the most likely explanation $E_{x,y}^*$ for the top-goal G_x , is formulated as (4) in D-PRISM and computed by dynamic programming just like PRISM.

$$\begin{aligned} E_{x,y}^* &= \operatorname{argmax}_{E_{x,y} \in \phi(G_x)} p(E_{x,y} \mid G_x) \\ &= \operatorname{argmax}_{E_{x,y} \in \phi(G_x)} \frac{1}{Z(G_x)} \exp \left(\sum_{i,v} \lambda_{i,v} \sigma_{i,v}(E_{x,y}) \right) \\ &= \operatorname{argmax}_{E_{x,y} \in \phi(G_x)} \sum_{i,v} \lambda_{i,v} \sigma_{i,v}(E_{x,y}) \end{aligned} \quad (4)$$

5 Experiments with three basic models

In this section, we conduct learning experiments with CRFs⁸. CRFs are encoded by D-PRISM programs while their generative counterparts are encoded by PRISM programs. We compare their accuracy in discriminative tasks. We consider three basic models, logistic regression, a linear-chain CRF and a CRF-CFG, and learn their parameters by L-BFGS.

5.1 Logistic regression with UCI datasets

We select four datasets with no missing data from the UCI Machine Learning Repository [6] and compare prediction accuracy, one by logistic regression written in D-PRISM and the other by a naive Bayes model (NB) written in PRISM.

We use the program in Fig. 1 with an appropriate modification of `values/2` declarations. The result by ten-fold cross-validation is shown in Table 1 with standard deviation in parentheses. Table 2 contains learning time for each dataset. We can see, except for the zoo dataset, logistic regression by D-PRISM outperforms naive Bayes by PRISM at the cost of considerably increased learning time for larger datasets⁹.

⁸ Experiments in this paper are done on a single machine with Core i7 Quad 2.67GHz×2 CPU and 72GB RAM running OpenSUSE 11.2.

⁹ In this paper, accuracies in bold letters indicate that they are the best performance and the difference is statistically significant by t-test at 0.05 significance level. Learning time is an average over five runs.

Table 1 Logistic-regression and naive Bayes : UCI datasets and accuracy

				D-PRISM	PRISM
Model				logistic-regression	naive Bayes
Dataset	Size	#Class	#Attr.		
zoo	101	7	16	96.00%(5.16)	97.0%(6.74)
car	1728	4	6	93.28% (2.02)	86.11%(1.47)
kr-vs-kp	3196	2	36	93.58% (4.40)	87.92%(1.69)
nursery	12960	5	8	92.54% (0.60)	90.27%(0.97)

Table 2 Logistic-regression and naive Bayes : Learning time (sec)

	D-PRISM	PRISM
Model	logistic-regression	naive Bayes
Method	L-BFGS	counting
zoo	0.09(0.00)	0.04(0.00)
car	0.04(0.00)	0.04(0.00)
kr-vs-kp	145.35(0.41)	0.30(0.00)
nursery	321.65(1.23)	0.38(0.00)

5.2 Linear-chain CRF with the Penn Treebank

We here compare a linear-chain CRF encoded by a D-PRISM program and an HMM encoded by a PRISM program using sequence data extracted from the Penn Treebank [16]. What we actually do is to write an HMM program in PRISM for complete data and another program for incomplete data and consider their union as a D-PRISM program defining a linear-chain CRF, similarly to the case of naive Bayes and logistic regression. For simplicity we employ default features, i.e. the count of various `msw` atoms in an explanation.

Fig. 2 is a sample D-PRISM program for a CRF with two states $\{s0, s1\}$ and two emission symbols $\{a, b\}$. `hmm0/2` describes complete data and corresponds to

```

% HMM specification                                % HMM specification for the Penn tree bank
%      for a sample HMM                            %
values(init,[s0,s1]).                               % values(init,[NNP,VBZ,s_dot,t_s_paren_1,...])
values(tr(_),[s0,s1]).                             % values(tr(_),[NNP,VBZ,s_dot,t_s_paren_1,...])
values(out(_),[a,b]).                               % values(out(_),[mss_dot,haag,plays,elianti,...])

hmm0([X0|Xs],[Y0|Ys]):- msw(init,Y0),msw(out(Y0),X0),hmm1(Y0,Xs,Ys).
hmm1(_,[ ],[ ]).
hmm1(Y0,[X|Xs],[Y|Ys]):- msw(tr(Y0),Y),msw(out(Y),X),hmm1(Y,Xs,Ys).

hmm0([X|Xs]):- msw(init,Y0),msw(out(Y0),X),hmm1(Y0,Xs).
hmm1(_,[ ]).
hmm1(Y0,[X|Xs]):- msw(tr(Y0),Y),msw(out(Y),X),hmm1(Y,Xs).

```

Fig. 2 Linear-chain CRF program

$G_{x,y}$ in (1) whereas `hmm0/1` is for incomplete data and corresponds to G_x in (1)¹⁰. As a CRF program, ground `msw` atoms such as `msw(init,s0)`, `msw(tr(s0),s1)` and `msw(out(s0,a))` represent binary feature functions over sequences of state transitions and emitted symbols. For example `msw(tr(s0),s1)` returns 1 (true) if the state transition sequence contains a transition from `s0` to `s1` else 0 (false).

We conduct a comparison of prediction accuracy by a linear-chain CRF and an HMM using the D-PRISM program in Fig. 2. The task is to predict the POS (Part Of Speech) tag sequence (hidden state sequence) for a given sentence (emitted symbol sequence). As learning data, we use two sets of pairs of sentence and POS tag sequence extracted from the Penn Treebank [16]: section-02 in the WSJ (Wall Street Journal articles) corpus referred to here as WSJ02-ALL and its subset referred to as WSJ02-15, consisting of data of length less-than or equal to 15. Their statistics are shown in Table 3.

Table 4 contains prediction accuracy (%) by eight-fold cross-validation and learning time taken for WSJ02-ALL. Parameters are learned by L-BFGS for D-PRISM and by counting for PRISM. The table clearly demonstrates again that we can achieve better prediction performance at the cost of increased learning time; D-PRISM gains 5.94% increase in prediction accuracy for the WSJ02-15 dataset but learning time by L-BFGS in D-PRISM is about 60 times longer than that by counting in PRISM.

Table 3 Penn Treebank data

Dataset	Size	Ave-len	#Tags	#Words
WSJ02-15	1087	9.69	40	3341
WSJ02-ALL	2419	19.28	45	8476

Table 4 Linear-chain CRF and HMM : Labeling accuracy and learning time

	D-PRISM	PRISM
Model	linear-chain CRF	HMM
Method	L-BFGS	counting
Accuracy (WSJ02-15)	83.17% (1.23)	77.23%(1.38)
(WSJ02-AL)	90.60% (0.32)	87.27%(0.29)
Learning time (sec) (WSJ02-15)	499.34(1.06)	8.04 (0.00)

5.3 CRF-CFG with the ATR tree corpus

We here deal with probabilistic grammars which graphical models are unable even to represent. We compare the parsing accuracy of a CRF-CFG described by a D-PRISM program and that of a PCFG described by a PRISM program. We do not use features other than the count of a rule in the parsing tree. To save space, we omit programs though they are (almost) identical.

¹⁰ Using “`hmm0([X0|Xs]):- hmm0([X0|Xs],_)`” to define `hmm0/1` is possible and theoretically correct but would kill the effect of tabling. This problem is discussed in Section 7.

As a dataset, we use the ATR tree corpus and its associated CFG [28]. Their statistics are shown in Table 5. After parameter learning by regularised conditional likelihood for the CRF-CFG and by the usual likelihood for the PCFG, we compare their parsing accuracy by ten-fold cross-validation. The task is to predict a parse tree given a sentence and the predicted tree is considered correct when it exactly coincides with the one for the sentence in the ATR tree corpus (exact match). As a reference, we also measure parsing accuracy by PCFG whose parameters are learned from incomplete data, i.e. sentences by the EM algorithm in PRISM.

Table 5 ATR corpus

Size	Ave-len	#Rules	#Nonterminals	#Terminals
10995	9.97	861	168	446

Table 6 CRF-CFG and PCFG : Parsing accuracy and learning time

	D-PRISM	PRISM	
Model	CRF-CFG	PCFG	
Method	L-BFGS	counting	EM
Accuracy	82.74% (1.62)	79.06%(1.25)	70.02%(0.87)
Learning time (sec)	205.51 (0.79)	2.30 (0.26)	65.72 (1.46)

Table 6 tells us that when a tree corpus is available, as reported in [5], shifting from PCFG (PRISM) to CRF-CFG (D-PRISM) yields much better prediction performance (and shifting cost is almost zero if we use D-PRISM) but at the same time this shifting incurs almost two orders of magnitude longer learning time.

6 Exploring new models

In this section, we demonstrate how the power of D-PRISM is exploited to explore new probabilistic models. We propose two new models. One is CRF-BNCs which are a CRF version of Bayesian networks classifiers. The other is CRF-LCGs which are a CRF version of probabilistic left-corner grammars that generatively formalize probabilistic left-corner parsing. We first introduce CRF-BNCs.

6.1 CRF-BNCs

Bayesian network classifiers (BNCs) [7, 2] are a generalization of naive Bayes classifiers. They use general Bayesian networks (BNs) as a classifier and allow dependencies among attributes unlike naive Bayes classifiers. Although BNCs outperform NBs classifiers in accuracy, they are still generative. We here introduce a CRF version of BNCs, *conditional random field BNCs* (CRF-BNCs), and empirically show that CRF-BNCs can outperform BNCs. Due to space limitations, we explain CRF-BNCs by an example.

CRF-BNCs are obtained, roughly speaking, by generalizing conditional probability tables in Bayesian networks to potential functions followed by normalization

w.r.t. the class variable¹¹. Fig. 3 is an example of Bayesian network for the car dataset in the UCI Machine Learning Repository [6]. It has a class variable C and six attribute variables, B , M , D , P , L and S . We assume they have dependencies designated in Fig. 3.

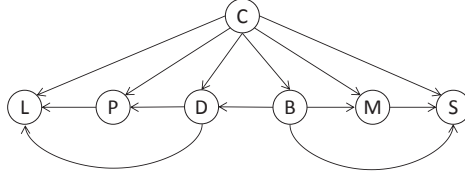


Fig. 3 Bayesian network for the car dataset

Implementing a CRF-BNC for Fig. 3 is easy in D-PRISM. We have only to write a usual generative Bayesian network program DB_1 in PRISM shown in Fig. 4 and run it as a D-PRISM program. In Fig. 4, the first clause about `bn` predicate defines an unnormalized probability $q_{DB_1}(\text{bn}(\text{Attrs}))$ and the second one defines $q_{DB_1}(\text{bn}(\text{Attrs}, C))$. So the conditional distribution $p(C \mid \text{Attrs})$ is computed as $q_{DB_1}(\text{bn}(\text{Attrs}, C)) / q_{DB_1}(\text{bn}(\text{Attrs}))$.

We conduct a learning experiment similarly to Section 5 to compare the CRF-BNC in Fig. 4 and its original BNC and obtain Table 7 for accuracy by ten-fold cross-validation and Table 8 for learning time of each dataset¹². Our experiment, though small, strongly suggests that when datasets are large enough, CRF-BNCs can outperform BNCs by a considerable margin at the cost of long learning time.

¹¹ Since CRF-BNCs preserve the graph structure of Bayesian networks, probabilistic inference by belief propagation can be efficiently carried for both of them with the same time complexity.

¹² Due to space limitations Bayesian networks for `zoo`, `kr-vs-kp` and `nursery` are omitted.

```

values(class, [unacc, acc, good, vgood]).
values(attr(buying, _), [vhigh, high, med, low]).
...
values(attr(safety, _), [low, med, high]).

bn(Attrs):- bn(Attrs, _).           % defines q(x) where x = Attrs
bn(Attrs, C):-                     % defines q(x,y) where x = Attrs, y = C
    Attrs = [B,M,D,P,L,S],
    msw(class, C), msw(attr(buying, [C]), B), msw(attr(maint, [B,C]), M),
    msw(attr(doors, [B,C]), D), msw(attr(persons, [D,C]), P),
    msw(attr(lug_boot, [D,P,C]), L), msw(attr(safety, [B,M,C]), S).

```

Fig. 4 CRF-BN program DB_1 for the car dataset

Table 7 CRF-BNC and BNC : UCI datasets and accuracy

				D-PRISM	PRISM
Model				CRF-BNC	BNC
Dataset	Size	#Class	#Attr.		
zoo	101	7	16	98.0 %(4.21)	98.09%(4.03)
car	1728	4	6	99.82% (0.54)	91.55%(1.92)
kr-vs-kp	3196	2	36	97.87% (0.85)	88.76%(1.31)
nursery	12960	5	8	96.57% (0.43)	92.46%(0.59)

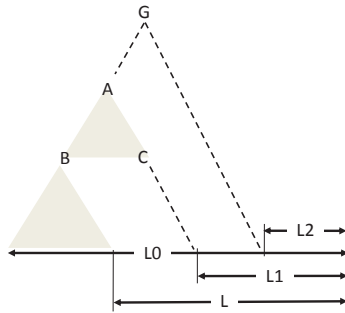
Table 8 CRF-BNC and BNC : Learning time (sec)

	D-PRISM	PRISM
Model	CRF-BNC	BNC
Method	L-BFGS	counting
zoo	0.12(0.00)	0.05(0.00)
car	0.92(0.00)	0.08(0.00)
kr-vs-kp	53.58(4.15)	0.34(0.00)
nursery	106.65(6.71)	0.42(0.00)

6.2 CRF-LCGs

A second new model class is *CRF-left-corner grammars* (CRF-LCGs). CRF-LCGs are a CRF version of probabilistic left-corner grammars (PLCGs) and considered dual to CRF-CFGs [5] in the sense that the former is based on bottom-up parsing, i.e. left-corner parsing [14,29] whereas the latter is based on top-down parsing. Although left-corner parsing is more context-dependent than top-down parsing and accordingly CRF-LCGs are expected to perform better than CRF-CFGs in parsing, no proposal of CRF-LCGs has been made yet to our knowledge.

Recall that left-corner parsing is procedurally defined through three parsing operations, i.e. shift, attach and project. However it can be defined logically by a pure logic program that describes various relationships among partial parse trees spanning substrings of the input sentence. Let N be a nonterminal and call a partial parse tree with root N N -tree. Fig. 5 is a snapshot of left-corner parsing when the

**Fig. 5** Partial parse trees constructed in left-corner parsing

```

plcg(L0):-
    start_symbol(C),
    g_call([C],L0,[]).

g_call([],L,L).
g_call([G|R],[Wd|L],L2):-
    ( terminal(G) ->
        G = Wd, L1 = L
    ; msw(first(G),Wd),
        lc_call(G,Wd,L,L1) ),
    g_call(R,L1,L2).

lc_call(G,B,L,L2):-
    msw(lc(G,B),rule(A,[B|RHS2])),
    g_call(RHS2,L,L1),
    ( G == A -> attach_or_project(A,Op),
        ( Op == attach, L2=L1
        ; Op == project, lc_call(G,A,L1,L2) )
    ; lc_call(G,A,L1,L2) ).
attach_or_project(A,Op):-
    ( reachable(A,A) -> msw(attach(A),Op) ; Op = attach ).

plcg(L0,T):-
    start_symbol(C),
    g_call_t([C],L0,[],[T]).

g_call_t([],L,L,[]).
g_call_t([G|R],[Wd|L],L2,T):-
    ( terminal(G) ->
        G = Wd, L1 = L, T = [Wd|TR]
    ; msw(first(G),Wd), T = [TG|TR],
        lc_call_t(G,Wd,L,L1,Wd,TG) ),
    g_call_t(R,L1,L2,TR).

...

```

Fig. 6 PLCG parsers for sentences (left) and for sentence-tree pairs (right)

B-tree is projected by a CFG rule $A \rightarrow B \ C$ to complete a G-tree where G and A are in the left-corner relation [14,15].

By translating the relationships that hold among various partial parse trees in Fig. 5 into a logic program, we obtain a bottom-up parser for probabilistic left-corner grammars as illustrated on the left in Fig. 6. There, for example, `lc_call(G,B,L,L2)` holds true for the parsing configuration described in Fig. 5 (details omitted)[14]. Similarly we write a parsing program in PRISM for complete data (sentence L0 and its tree T) placed on the right in Fig. 6, which is almost isomorphic to the left program. The left and right PRISM programs combined together constitute a D-PRISM program for CRF-LCGs (values declarations that specify CFG rules are not shown).

Following the case of CRF-CFG and PCFG in Section 5, we measure the parsing accuracy by ten-fold cross-validation of CRF-LCG and PLCG for the ATR corpus and the associated CFG using the D-PRISM program in Fig. 6. The result is shown in Table 9. CRF-LCG achieves the highest parsing accuracy compared to PLCG, PCFG and CRF-CFG but again at the cost of long learning time.

Table 9 CRF-LCG and PLCG : Parsing accuracy and learning time

	D-PRISM	PRISM	
Model	CRF-LCG	PLCG	
Method	L-BFGS	counting	EM
Accuracy	87.26% (0.99)	82.70%(1.97)	72.45%(1.37)
Learning time (sec)	290.89 (1.86)	9.41 (0.15)	102.24 (8.61)

7 Program transformation for incomplete data

As explained in Section 4, in D-PRISM the user needs to define two top-goals, $G_{x,y}$ for complete data (x, y) and G_x for incomplete data x , each defining unnormalized distributions $q_{DB}(G_{x,y})$ and $q_{DB}(G_x)$ respectively. Then $p(y \mid x)$ is computed as $\frac{q_{DB}(G_{x,y})}{q_{DB}(G_x)}$. However since $G_{x,y}$ and G_x are logically connected as $G_x \Leftrightarrow \exists y G_{x,y}$, it is theoretically enough and correct to add a clause “ $\mathbf{g}(\mathbf{X}) : \neg \mathbf{g}(\mathbf{X}, \mathbf{Y})$ ” to the program for $G_{x,y}$ to obtain a program for $q_{DB}(G_x)$ ¹³. This is what we did for the naive Bayes program in Fig. 1.

Unfortunately this simple approach does not work in general. The reason is that the search for all explanations for $\mathbf{g}(\mathbf{X})$ causes an exhaustive search for proofs of $\mathbf{g}(\mathbf{X}, \mathbf{Y})$ for all possible values of \mathbf{Y} . Consequently when a subgoal occurring in the search process that carries \mathbf{Y} is proved with some value $\mathbf{Y} = a$ and tabled, i.e. stored in the memory for reuse, it has little chance of being reused later because \mathbf{Y} in the subgoal mostly takes different values from a . As a result the effect of tabling is almost nullified, causing an exponential search time for all explanations for $\mathbf{g}(\mathbf{X})$.

To avoid this negative effect of the redundant argument, \mathbf{Y} , we often have to write a specialized program for $\mathbf{g}(\mathbf{X})$, independently of a program for $\mathbf{g}(\mathbf{X}, \mathbf{Y})$, that does not refer to \mathbf{Y} ; in the case of linear-chain CRF program in Fig. 2, we wrote two programs, one for $\text{hmm0}(\mathbf{X}, \mathbf{Y})$ (complete data) and the other for $\text{hmm0}(\mathbf{X})$ (incomplete data). The latter is a usual HMM program and efficient tabling is possible that guarantees linear time search for all explanations. However, writing a specialized program for $\mathbf{g}(\mathbf{X})$ invites another problem of program correctness. When we independently write two programs, DB_1 for $\mathbf{g}(\mathbf{X}, \mathbf{Y})$ and DB_2 for $\mathbf{g}(\mathbf{X})$, they do not necessarily satisfy $q_{DB_1}(\exists \mathbf{X} \mathbf{g}(\mathbf{X}, \mathbf{Y})) = q_{DB_2}(\mathbf{g}(\mathbf{X}))$ which is required for sound computation of the conditional distribution of $p(y \mid x)$. It is therefore hoped to find a way of obtaining DB_2 satisfying this property.

¹³ Here it is assumed that “ $\mathbf{g}(\mathbf{X}, \mathbf{Y})$ ” is a top-goal for $G_{x,y}$ and “ $\mathbf{g}(\mathbf{X})$ ” for G_x respectively.

```

(1) hmm0([X0|Xs], [Y0|Ys]) :- msw(init, Y0), msw(out(Y0), X0), hmm1(Y0, Xs, Ys).
(2) hmm1(_, [], []).
(3) hmm1(Y0, [X|Xs], [Y|Ys]) :- msw(tr(Y0), Y), msw(out(Y), X), hmm1(Y, Xs, Ys).
(4) hmm0(X) :- hmm0(X, Y).
(5) hmm1(Y0, Xs) :- hmm1(Y0, Xs, Ys).

(6) hmm0([X0|Xs]) :- msw(init, Y0), msw(out(Y0), X0), hmm1(Y0, Xs, Ys).
    -- from unfolding (4) by (1)
(7) hmm0([X0|Xs]) :- msw(init, Y0), msw(out(Y0), X0), hmm1(Y0, Xs).
    -- from folding (6) by (5)
(8) hmm1(Y0, []).
    -- unfolding (5) by (2) and (3) giving (8) and (9)
(9) hmm1(Y0, [X|Xs]) :- msw(tr(Y0), Y), msw(out(Y), X), hmm1(Y, Xs, Ys).
(10) hmm1(Y0, [X|Xs]) :- msw(tr(Y0), Y), msw(out(Y), X), hmm1(Y, Xs).
    -- from folding (9) by (5)

```

Fig. 7 Unfold/fold program transformation of `hmm0/1`

One way to achieve this is to use meaning preserving unfold/fold transformation for logic programs [27, 20]. It is a system of program transformation containing rules for unfolding and folding operations. Unfolding replaces a goal with the matched body of a clause whose head unifies with the goal and folding is a reverse operation. There are conditions on transformation that must be met to ensure that the transformation is meaning preserving, i.e. the least model of programs is preserved through transformation (see [27, 20] for details). Note that meaning preserving unfold/fold program transformation also preserves the set of all explanations for a goal. So if DB_2 is obtained from $DB_1 \cup \{g(X) :- g(X, Y)\}$ by such transformation, both programs have the same set of all explanations for $g(X)$, and hence the desired property $q_{DB_1}(\exists X g(X, Y)) = q_{DB_2}(g(X))$ holds. In addition, usually, DB_2 does not refer to the cumbersome Y .

Fig. 7 illustrates a process of such program transformation. It derives an HMM program for $hmm0(X)$ computing incomplete data from a program for $hmm0(X, Y)$ computing complete data using a transformation system described in [27]. It starts with the initial program defining $hmm0(X, Y)$ consisting of $\{(1), (2), (3)\}$ together with two defining clauses for new predicates, i.e. (4) for $hmm0(X)$ and (5) for $hmm1(Y0, Xs)$. The transformation process begins by unfolding the body goal $hmm0(X, Y)$ in (4) by (1) and folding $hmm1(Y0, Xs, Ys)$ by (5) follows, resulting in (7). The defining clause (5) for $hmm1(Y0, Xs)$ is processed similarly. The final program obtained is $\{(7), (8), (10)\}$ which coincides with the HMM program for $hmm0(X)$ in Fig. 2.

This example exemplifies that unfold/fold transformation has the power of eliminating the redundant argument Y in $g(X) :- g(X, Y)$ and deriving a specialized program for $g(X)$ that does not refer to Y and hence is suitable for tabling. However how far this transformation is generally applicable and how far it can be automated is future work.

8 Discussion and future work

There are already discriminative modeling languages for CRFs such as Alchemy [10] based on MLNs and Factorie [17] based on factor graphs. To define potential functions and hence models, the former uses weighted clauses whereas the latter uses imperatively defined factor graphs. Both use Markov chain Monte-Carlo (MCMC) for probabilistic inference. D-PRISM differs from them in that although programs are used to define CRFs like MLNs and Factorie, they are purely generative, computing output from input, and probabilities are computed exactly by dynamic programming. TildeCRF [8] learns CRFs over sequences of ground atoms. Potential functions are computed by weighted sums of relational regression trees applied to an input sequence with a fixed size window. TildeCRF is purely discriminative and unlike D-PRISM uses a fixed type of potential function. Also it is designed for linear-chain CRFs and more complex CRFs such as CRF-CFGs are not intended or implemented.

D-PRISM is interesting from the viewpoint of statistical machine learning in that it builds discriminative models from generative models and offers a general approach to implementing generative-discriminative pairs. This unique feature also makes it relatively easy and smooth to develop new discriminative models from generative models as we demonstrated in Section 6. In addition, as is shown by

every experiment in this paper, there is a clear trade-off between accuracy (by discriminative models) and learning time (by generative models), and hence we have to choose which type of model to use, depending on our purpose. D-PRISM assists our choice by providing a unified environment to test both types.

Compared to PRISM, D-PRISM has no restriction on programs such as the uniqueness condition, exclusiveness condition and independence condition [23]. Consequently non-exclusive or is permitted in a program. Also probability computation is allowed to fail by constraints. For example it is straightforward to add linguistic constraints such as subject-verb agreement to a PCFG by adding an extra argument carrying such agreement information to the program. Although loss of probability mass occurs due to disagreement in the generating process, normalization recovers a distribution and we obtain a constraint CRF-CFG as a result. Of course this freedom is realized at the cost of normalization which may be prohibitive even when dynamic programming is possible. This would happen when adding too many constraints, e.g., agreement in number, gender, tense and so on to a PCFG. Thanks to the removal of restrictive conditions however, D-PRISM is now more amenable to structure learning in ILP than PRISM, which is expected to open up a new line of research of learning CRFs in ILP.

In this paper we concentrated on learning from complete data in CRFs and missing value is not considered. When there are missing values, for example when some labels on a sequence in a linear-chain CRF are missing, the data is incomplete and parameter learning becomes much harder, if not impossible. There is a method of parameter learning from incomplete data for conditional distributions using EM. It is developed for PRISM programs with failure [24] and learns parameters from a conditional distribution of the form $p_{DB}(G_x \mid \text{success})$ where $\text{success} = \exists x G_x$ and G_x is a goal for incomplete data x that may fail. The point in [24] is to automatically synthesize **failure** predicate such that $p_{DB}(\text{success}) = 1 - p_{DB}(\text{failure})$ and rewrite the conditional distribution as an infinite series $p_{DB}(G_x \mid \text{success}) = p_{DB}(G_x)(1 + p_{DB}(\text{failure}) + p_{DB}(\text{failure})^2 + \dots)$ to which EM is applicable (the FAM algorithm [4]). Although whether the adaptation of this technique to EM learning of CRFs with incomplete data is possible or not is unknown, it seems worth pursuing considering the simplicity of EM compared to complicated gradient-based parameter learning algorithms for incomplete data.

In Section 7, the unfold/fold program transformation is used to remove the redundant argument Y from $\text{hmm0}(X, Y)$. Y is a non-discriminating argument in the sense of [3]. Christiansen and Gallagher gave a deterministic algorithm to eliminate such non-discriminating arguments without affecting the program's runtime behavior [3]. Actually deleting non-discriminating arguments from clauses for $\text{hmm0}(X, Y)$ in Fig. 2 results in the same HMM program obtained by program transformation. Compared to their approach however, our approach is based on non-deterministic unfold/fold program transformation and allows for an introduction of new predicates. Clarifying the relationship between these two approaches is future work.

Currently only binary features or their counts are allowed in D-PRISM. Introducing real-valued features is also a future work and so is a mechanism of parameter

tying. Finally, D-PRISM is experimentally implemented at the moment and we hope it is part of the PRISM package in the future.

9 Conclusion

We have introduced D-PRISM, a logic-based generative language for discriminative modeling. As examples show, D-PRISM programs are just PRISM programs with probabilities replaced by weights. It is the first modeling language to our knowledge that generatively defines CRFs and their extensions to probabilistic grammars. We can freely build logistic regression, linear-chain CRFs, CRF-CFGs or new models generatively with almost the same modeling cost as PRISM while achieving better performance in discriminative tasks.

References

1. Biba, M., Xhafa, F., Esposito, F., Ferilli, S.: Stochastic simulation and modelling of metabolic networks in a machine learning framework. *Simulation Modelling Practice and Theory* **19**(9), 1957–1966 (2011)
2. Cheng, J., Greiner, R.: Comparing bayesian network classifiers. In: *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence (bUAI'99)*, pp. 101–108 (1999)
3. Christiansen, H., Gallagher, J.: Non-discriminating arguments and their uses. In: *Proceedings of the 25th International Conference on Logic Programming (ICLP'09)*, LNCS 5649, pp. 55–69 (2009)
4. Cussens, J.: Parameter estimation in stochastic logic programs. *Machine Learning* **44**(3), 245–271 (2001)
5. Finkel, J., Kleeman, A., Manning, C.: Efficient, feature-based, conditional random field parsing. In: *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (ACL'08)*, pp. 959–967 (2008)
6. Frank, A., Asuncion, A.: UCI machine learning repository (2010). URL <http://archive.ics.uci.edu/ml>
7. Friedman, N., Geiger, D., Goldszmidt, M.: Bayesian network classifiers. *Machine learning* **29**(2), 131–163 (1997)
8. Gutmann, B., Kersting, K.: TildeCRF: Conditional random fields for logical sequences. In: *Proceedings of the 15th European Conference on Machine Learning (ECML-06)*, pp. 174–185 (2006)
9. Johnson, M.: Joint and conditional estimation of tagging and parsing models. In: *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL-01)*, pp. 322–329 (2001)
10. Kok, S. and Singla, P. and Richardson, M. and Domingos, P.: The Alchemy system for statistical relational AI. Technical report, Department of Computer Science and Engineering, University of Washington (2005). URL <http://www.cs.washington.edu/ai/alchemy>
11. Lafferty, J., McCallum, A., Pereira, F.: Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: *Proceedings of the 18th International Conference on Machine Learning (ICML'01)*, pp. 282–289 (2001)
12. Liang, P., Jordan, M.: An asymptotic analysis of generative, discriminative, and pseudo-likelihood estimators. In: *Proceedings of the 25th international conference on Machine learning (ICML'08)*, pp. 584–591 (2008)
13. Liu, D., Nocedal, J.: On the limited memory BFGS method for large scale optimization. *Mathematical Programming* **45**, 503–528 (1989)
14. Manning, C.: Probabilistic parsing using left corner language models. In: *Proceedings of the 5th International Conference on Parsing Technologies (IWPT-97)*, pp. 147–158. MIT Press (1997)
15. Manning, C.D., Schütze, H.: *Foundations of Statistical Natural Language Processing*. The MIT Press (1999)

16. Marcus, M., Santorini, B., Marcinkiewicz, M.: Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics* **19**, 313–330 (1993)
17. McCallum, A., Schultz, K., Singh, S.: Factorie: Probabilistic programming via imperatively defined factor graphs. In: *Advances in Neural Information Processing Systems* 22, pp. 1249–1257 (2009)
18. Mørk, S. and Holmes, I.: Evaluating bacterial gene-finding HMM structures as probabilistic logic programs. *Bioinformatics* **28**(5), 636–642 (2012)
19. Ng, A., Jordan, M.: On Discriminative vs. Generative Classifiers: A comparison of logistic regression and naive Bayes. In: *NIPS*, pp. 841–848 (2001)
20. Pettorossi, A., Proietti, M.: Transformation of logic programs: Foundations and techniques. *Journal of Logic Programming* **19,20**, 261–320 (1994)
21. Richardson, M., Domingos, P.: Markov logic networks. *Machine Learning* **62**, 107–136 (2006)
22. Sato, T., Kameya, Y.: PRISM: a language for symbolic-statistical modeling. In: *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI’97)*, pp. 1330–1335 (1997)
23. Sato, T., Kameya, Y.: Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research* **15**, 391–454 (2001)
24. Sato, T., Kameya, Y., Zhou, N.F.: Generative modeling with failure in PRISM. In: *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI’05)*, pp. 847–852 (2005)
25. Sneyers, J., Vennekens, J., De Schreye, D.: Probabilistic-logical modeling of music. In: *Proceedings of the 8th International Symposium on Practical Aspects of Declarative Languages (PADL’06)*, vol.3819, LNCS, pp. 60–72 (2006)
26. Sutton, C., McCallum, A.: An introduction to conditional random fields. *Foundations and Trends in Machine Learning* **4**(4), 267–373 (2012)
27. Tamaki, H., Sato, T.: Unfold/fold transformation of logic programs. In: *Proceedings of the 2nd International Conference on Logic Programming (ICLP’84)*, Lecture Notes in Computer Science, pp. 127–138. Springer (1984)
28. Uratani, N., Takezawa, T., Matsuo, H., Morita, C.: ATR integrated speech and language database. Technical Report TR-IT-0056, ATR Interpreting Telecommunications Research Laboratories (1994)
29. Van Uytsel, D., Van Compernelle, D., Wambacq, P.: Maximum-likelihood training of the PLCG-based language model. In: *Proceedings of the IEEE Automatic Speech Recognition and Understanding Workshop 2001 (ASRU’01)*, pp. 210–213 (2001)